6-1993

A Generalized Data Conversion System

Kevan Morgan
Utah State University

A Generalized Data Conversion System

Senior Project Report / Honors Thesis
Electrical Engineering Department / Honors Program
Utah State University

Kevan Morgan

June 2, 1993

Abstract

To assist in the processing of data files collected by the Space Dynamics Laboratory at Utah State University, the software project GENDACS has been completed. GENDACS, acronym for A Generalized Data Conversion System, will contribute to its research program.

Data are collected from various sensory instruments by the Computer Data Acquisition System. The instrument readings are brought in, tagged with the time they were taken, and stored on an optical disk in a compact, binary (non-ASCII) format. GENDACS is designed so that the program can read in such files, sort out the data, and convert them into a new ASCII configuration. The converted data are written to a separate file.

GENDACS provides considerable versatility in reading and writing different file configurations. This versatility is necessary because different data collecting software packages may organize and store data in varying styles. GENDACS makes it possible for an engineer to extract and manipulate data from different types of files, enabling other commercial graphing programs to analyze and display the results.

When the program is first operating, the user must define how the input data file is configured. Different input files may have different sequences of various data types. These data sequences need to be described to enable GENDACS to read in the data. Next, the user may give the program instructions to manipulate the data in various ways. Data values may be added or subtracted from each other, or constant numerical scaling factors may be applied to the data. Finally, the user must also define how the resultant data should be written out. Once this configuration information is given, GENDACS can proceed to process the file. Preliminary testing of GENDACS has confirmed its success in correctly processing data.

This software was written using object-oriented Pascal (programming language). The code is designed for use on a PC-based computer system. Although primarily written to facilitate data analysis being accomplished at the Space Dynamics Laboratory, this program could easily be adapted for other users as well.

Table of Contents

List of Figures

I. INTRODUCTION

The Space Dynamics Laboratory at Utah State University collects data on atmospheric emissions and associated geophysical phenomena that occur, in part, during auroral displays. These data are in a compact, binary format that must be processed into a more readable form before they can be analyzed. In response to this need, the program GENDACS was written.

The most important design goal was to make GENDACS versatile enough to allow the user, at the time the program is run, to specify the kind of file it will read in for processing. Any file in the MS-DOS environment that uses standard data types (i.e. 8, 16, or 32-bit signed or unsigned integer values or 48-bit floating point values) can be read in. The data, once read in, can be manipulated and written out in a wide variety of configurations, again depending on the instructions the user gives the program. The newly formatted data can then be entered into other commercial data analysis programs for further graphical and statistical review.

II. PROJECT BACKGROUND

A. *History*

The Space Dynamics Laboratory at Utah State University (SDL/USU or SDL) is involved in research on upper atmospheric processes (for example, auroral displays). Instruments used include magnetometers, scanning photometers, and riometers. Magnetometers record magnetic activity, photometers record light intensity at particular wavelengths, and riometers record atmospheric absorption occurring at certain radio frequencies. SDL currently uses the Computer Data Acquisition System (CDAS) to collect data from these instruments. All collected data have been stored in a compact, binary (non-ASCII) manner.

B. *Significance*

Because of the way the data were stored, additional processing of the data files was necessary before any useful evaluation can be performed. Although software existed that could help in the processing of these data files, it became apparent that a more powerful data formatting tool was needed. The software at the time was seriously limited in its scope and versatility, since only select types of data files could be read in. Furthermore, only fixed output formats for the processed data were available. To get the data into the final format desired, additional manipulation of the data often had to be performed by hand. This was very time-consuming and not very efficient. In addition, if SDL ever changes its data analysis techniques, it may require the data to be converted into other alternative formats. SDL needed a much more dynamic software package.

Beyond its own research efforts, SDL sometimes offers these data files to other institutions. These other institutions may need the data in a specific format that the previous software could not supply. This project set out to alleviate these concerns.

C. *Solution*

The program GENDACS was written to be a powerful, yet user-friendly, data formatting package for SDL. Although the program was directed towards CDAS-generated data, it has the versatility to address data files configured in other fixed, binary fashions. The user needs to know only how the data in the input file are organized. The user can then instruct GENDACS how the input file is configured, what conversion steps to perform on the data, and how the resultant data should be stored.

The current version of GENDACS does carry one limitation. It can only deal with input data files that have one or more repeated blocks of binary data. Bit for bit, it is assumed that the input file contains data stored in a fixed binary fashion. Files with data stored in a variable-length ASCII form cannot be processed.

III. DESIGN AND IMPLEMENTATION

A. *System Level Design*

1. *Program Flow*

A simple flow diagram, Figure 1, shows how the program operates. It describes the fundamental information needed for the data to be formatted correctly.

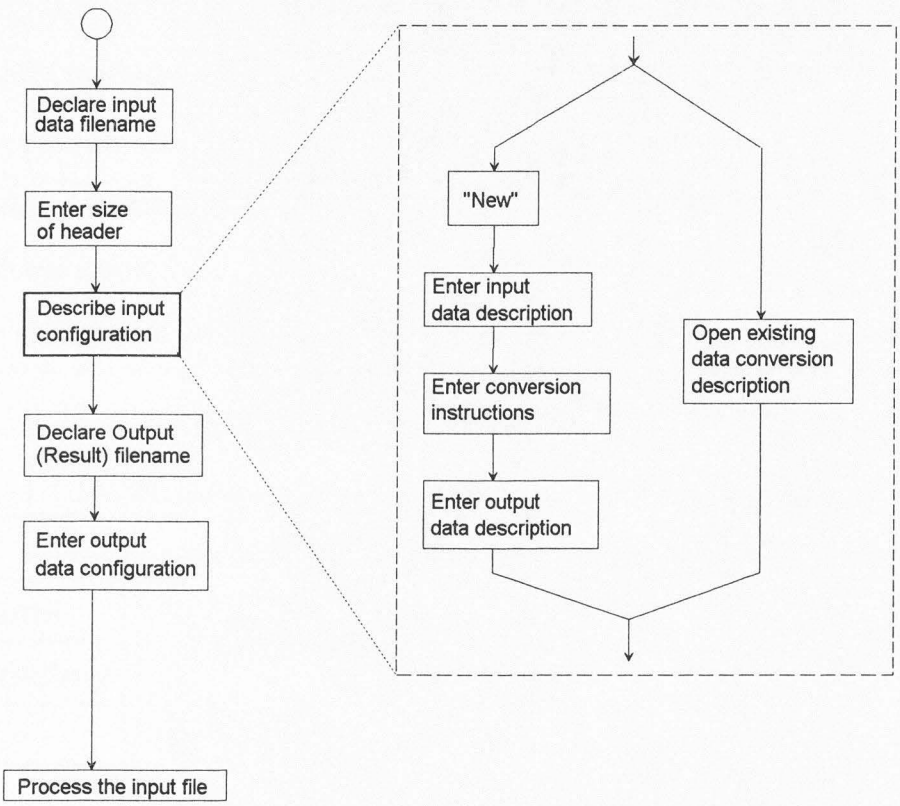

Figure 1. System Level Flow Diagram.

The user must supply the name of the input data file to be read in and the size, in bytes, of input header information to skip. If the input file has no header information, this field is left zero. The inset to the right (dotted-lines) gives greater detail to the procedure of describing the input file configuration. If the organization of the input file has been previously described, the user can

load in that description. If not, that information must be supplied. Further discussion of that process will be given later.

Since GENDACS writes out resultant data to a file and needs to know what to call it, an output file name must also be given. Configuring the output file will be also be discussed later in this report.

2. *Menu Structure*

GENDACS implements a DOS-based windowing user interface. Windows, dialog boxes, and information prompts are all included. These windows can be scrolled and moved around the screen. Like many software packages that use standard window interfaces, GENDACS has a pull-down menu bar across the top of the screen. This menu structure assists the user in entering the information necessary to process data files. See Figure 2.

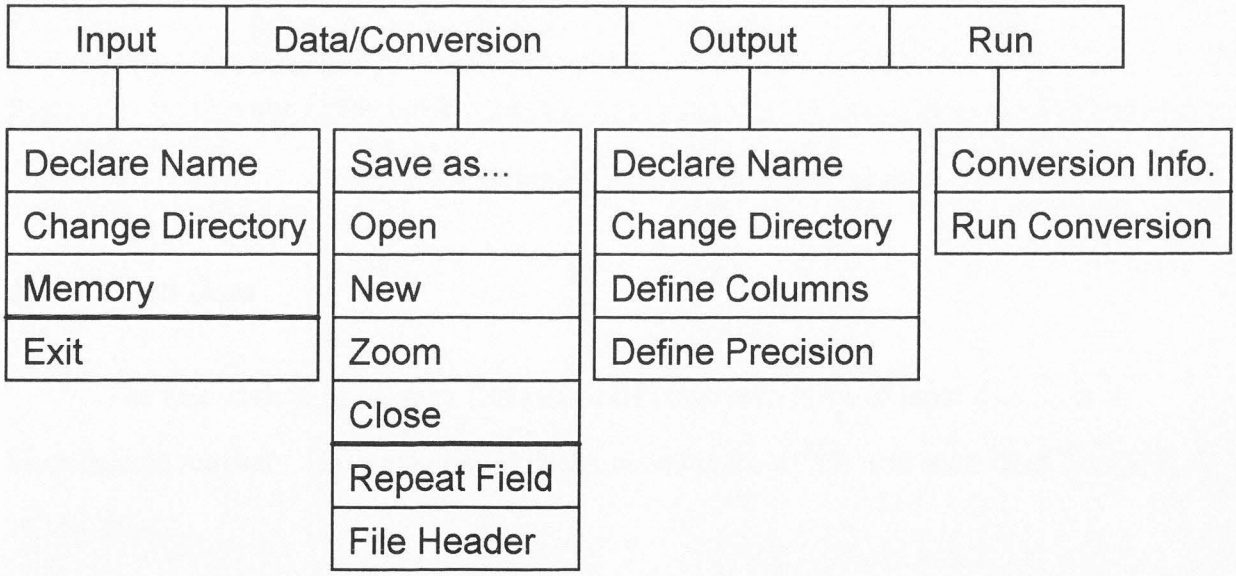

Figure 2. Menu Structure

3. *Conversion Description*

As GENDACS is running, a central window contains the user's description of the input data file as well as the conversion instructions. In this window, the user also partially declares how the resultant data should be written out. Figure 3 shows a sample data conversion window. Each line in the window details the processing of each input datum. If the user has already entered this descriptive information, it can be saved on disk for future data processing. Then, the user need only load in this previously stored information. Otherwise, the user selects "New" (see Figure 1) and begins to enter a new conversion description.

```
IN:Field#     Type        Conversion Instructions    OUT:Field#     Type
1.............word...........>.............................1..........real
2.............word.........../60;+o1.....................2..........real
3.............word.........../3600;+o1...................3..........real
4.............word...........x.....................................
5.............word...........+i4;/2.....................4..........real
6.............longinteger....*.00488...................5..........real
7.............byte...........x.....................................
8.............integer........*i7.......................6..........real
9.............shortinteger...x.....................................
```

Figure 3. Sample Conversion Information.

a. *Input Data*

The first column of Figure 3 (`IN:Field#`) tags each piece of input data with an identification number. The numbering is linear, counting from "`1`"; and each input data field must be identified.

In the second column of Figure 3, a description of the data type of each field of input data was entered. Differing data types have different characteristics. Figure 4 describes some of these differences.

Data Type	Size in Bytes	Range	Signed
integer	16-bits (2 bytes)	-32768...32767	Yes
byte	8-bits (1 byte)	0...255	No
short integer	8-bits (1 byte)	-128...127	Yes
word	16-bits (2 bytes)	0...65535	No
long integer	32-bits (4 bytes)	-2147483648...2147483647	Yes
real	48-bits (6 bytes)	2.9×10^{-39} ... 1.7×10^{38}	Yes

Figure 4. Data Types Available

There are three other floating-point data types that will be made available in future updates of GENDACS. These include "single" (4 bytes), "double" (8 bytes), and "extended" (10 bytes), each type having a varying degree of precision.

For example, where the first line of Figure 3 indicates that the input data type is "word," GENDACS will go out to the file and, at that point, access the next 16 bits of information. If those 16 bits do not hold a valid unsigned 16-bit value (a "word"), the data acquired will likely be garbage. For that reason, it is imperative that the user already know how the data were originally stored and to instruct the program accordingly. GENDACS follows only the directions given to it.

b. *Conversion Instructions*

The third column of Figure 3 lists some of the available conversion instructions. This is the most cryptic part of GENDACS. Two competing design goals helped shape the conversion instruction set and its syntax. On one hand, the instructions needed to be kept simple and easy to use. However, versatility in the number of ways to format data remained the main purpose of the project. After a significant amount of deliberation, a final set of instructions was drafted. The appendix offers a complete listing of the available instructions.

Basically, the user can take any piece of input data, output data, or immediate numerical value and, using these instructions, add, subtract, multiply, or divide them to create a desired result. For example, line 8 in Figure 3 says to take input data #8 and multiply it by input data #7 ("`*i7`"). Input data #7 could be a calibration or gain value and, thus, the result could be an adjusted or scaled value of #8. GENDACS allows the user to tailor a data formatting system for any data analysis scheme.

c. *Output*

The fourth and fifth columns of Figure 3 describe how the output data should be stored. They follow the same pattern as the input data description. The user tags each resultant value with an identification (field) number. However, the sequential order of the resultant data can be changed. The output ID numbers may be shifted around as needed. The user also identifies the data type(s) in which the data should be written out (i.e., as one of the types listed in Figure 4). Responsibility rests with the user to make certain the output data remain within the numerical ranges given by the specified data types.

After receiving all the necessary input, output, and conversion information, GENDACS is ready to process almost any data file the user has collected. Once again, the only restrictions are that the files contain data of the types that GENDACS can process and that it be stored in a fixed, binary sequence.

B. *Coding Language*

Another significant aspect of this project dealt with the language in which it was written. A new advance in programming style has provided engineers with a structured and extensible programming technique called Object-Oriented Programming (OOP). GENDACS was written in Object-Oriented Pascal using Turbo Pascal 6.0 (by Borland International Inc.). Without going

into deep discussion of OOP philosophy, it is appropriate to say that programs written using OOP techniques are much more modular and much easier to maintain than programs using standard techniques, a characteristic especially fitting for this project. As stated before, the main goal of this project was to preserve flexibility in processing all types of data. If new data types are developed, OOP style allows for extending and updating original source code without actually changing it. The engineer need only *redefine* and add onto the original program procedures to make them fit new specifications.

Turbo Pascal 6.0 also provides an OOP-based user interface framework called Turbo Vision. The objects (or structures) defined within Turbo Vision offer a programmer a "skeleton" around which to build a program. Since a great amount of time for any new software project is spent in creating a good user interface, Turbo Vision became an important part of GENDACS. Having ready-made support for creating menus, windows, and information boxes allowed for more time to be spent in designing and implementing the actual data conversion process.

Turbo Vision also offers support for "event driven" programming. When the user gives input to the program, an "event" is created. These packaged input events are sent throughout the program, looking for a procedure to "handle", or react, to them. The engineer, when programming in this style, does not need to be concerned about how the events are *created*, only how the events are *handled*. Each routine needs to know how to react to any input. For a more in-depth discussion of OOP, Turbo Vision, and event driven programming, see the Turbo Vision Guide or the Users' Guide provided with Turbo Pascal 6.0.

C. *Implementation -- Data Structures*

1. *Conversion Description Object*

When GENDACS is initially run, it does not know what kind of data files it will be reading; therefore, internal data structures to hold the incoming data are created only when the actual conversion is run. By the time the conversion is running, the user has entered a full

description of the input and output data configurations. Because the data structures are set up dynamically, a "linked list" structure became the optimal choice. Linked lists are composed of data records that include the memory address of the next data record. Set up like a chain, a linked list of data can get larger or smaller by simply adding or taking away a "link" (data record). Linked lists take up only as much computer memory as is needed at any instant.

The first linked list created by GENDACS holds the conversion description information entered by the user. For reference, Figure 3 shows a sample of that type of information. Each record (link) in this list is set up as seen in Figure 5a.

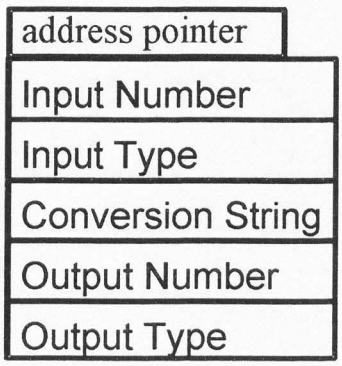

Figure 5a. Conversion Description Record

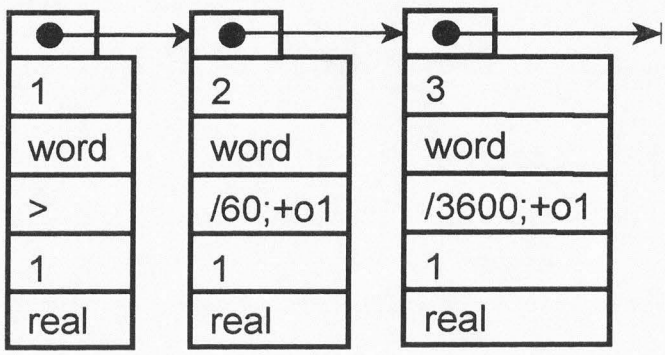

Figure 5b. Sample Linked List of Records

For each piece of input data, a conversion record is created and added to the list. Compare Figure 5b with the first three lines of Figure 3.

2. *Input/Output Data Object*

Once the data conversion process has begun, two other linked lists are created. These data structures hold actual data values, either read in from the input file or the resultant data to be stored out in the output file. Figures 6a and 6b show how these linked lists were designed.

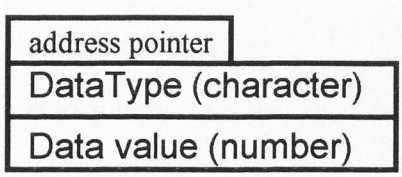

Figure 6a. Input/Output Data Record

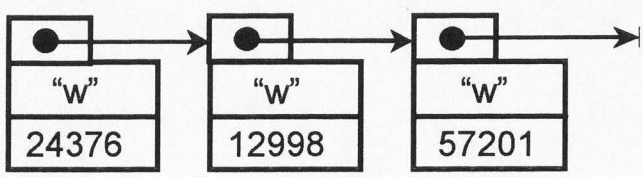

Figure 6b. Sample Data Linked List.

These lists can hold data of any of the types listed in Figure 4, namely, word, real, integer, byte, longinteger, and shortinteger. It is the user's description of the data that determines how these lists are set up.

3. *Conversion String*

The final data structure constructed by GENDACS holds the user-provided conversion instructions. (Again, for reference, see Figure 3.) Attempting to keep the user interface as simple as possible, the instruction codes given by the user are on a higher level than the codes GENDACS uses. This situation would be comparable to instructing a computer in Pascal (a higher level language) as opposed to lower level assembly language. When the conversion process is initiated, GENDACS takes all the conversion instructions given by the user and "compiles" them into a lower level, one dimensional string of characters. These characters are used by GENDACS to process the data. Because the number of conversion instructions is unknown when the program is first run, GENDACS again uses another dynamically-created linked list structure. This structure is much less complex than the other structures that hold the data. It is made up of 255-character strings, each with an address attached that can lead the program to the next 255-character string.

D. *Memory Issues*

After seeing so many data structures being created, any software engineer will logically question how efficiently the program uses computer memory. Computers do not have unlimited resources of memory. GENDACS, though far from perfect, appears to manage its usage of memory very well. Generally, when data files are created by the measuring instruments out in the field, the data are stored in small, repetitive blocks. Each block corresponds to the instant when the instruments make a measurement. GENDACS reads in each block, but only one block at a

time. When finished processing an individual block of input data, GENDACS discards it and reads in the next block. In this manner, it is able to process files that may be several megabytes in size without requiring enormous amounts of computer memory.

Memory restrictions arise when individual blocks of data become very large. When processing each block of data, GENDACS needs to read it in completely`. GENDACS also creates a companion data structure to hold the processed data. When combining the structures for the input and output data along with the structure that holds the conversion instructions, GENDACS may require more than the 640 KBytes of memory that MS-DOS allows. In that case, GENDACS cannot be used. However, those instances appear to be few in number. Unless several, high precision field instruments are connected together and their measurements are being combined into one block, most blocks of data within a data file will not be too large for GENDACS to handle.

IV. RESULTS

A. *Approach to Testing*

To test the validity of GENDACS-processed data, sample files of pseudo-data were constructed. Routines in assembly language were used to write out specific values to create test files of compact, binary data. Some of these blocks of "data" were appended to each other in order to simulate larger data files consisting of multiple blocks of data. Finally, a large scale test of GENDACS was carried out on actual data files collected by the Space Dynamics Laboratory.

B. *Sample Data Conversion*

1. *Straight-Across Processing*

The first test involved simply extracting binary data from a file and writing it out in ASCII form. Figure 7 shows conversion instructions that would implement this.

```
IN:Field#      Type          Conversion Instructions      OUT:Field#      Type
 1.............integer........>.............................1........integer
 2.............real...........>.............................2........real
 3.............shortint.......>.............................3........shortint
 4.............byte...........>.............................4........byte
 5.............longint........>.............................5........longint
 6.............word...........>.............................6........word
```

Figure 7. Straight-Across Processing.

The instruction ">" means to pass the data unaltered along to the output.

2. *Data Filtering*

The second test involved extracting only selected data values. The data selected was again written out unaltered. Figure 8 shows a sample of this processing application.

```
IN:Field#      Type          Conversion Instructions      OUT:Field#      Type
 1.............integer........>.............................1........integer
 2.............real...........x.........................................
 3.............shortint.......>.............................2........shortint
 4.............byte...........x.........................................
 5.............longint........>.............................3........longint
 6.............word...........x.........................................
```

Figure 8. Data Filtering.

The instruction "x" means to ignore that data value.

3. *Data Averaging*

The third test began to use more of the conversion instructions GENDACS provides. The aim of this test was to show how averaging of data could be implemented. See Figure 9.

```
IN:Field#      Type          Conversion Instructions      OUT:Field#      Type
 1.............integer........x.........................................
 2.............real...........+i1;/2.....................1........real
 3.............shortint.......x.........................................
 4.............byte...........+i3;/2.....................2........real
 5.............longint........x.........................................
 6.............word...........+i5;/2.....................3........real
```

Figure 9. Data Averaging.

4. *Changing Order and Type*

Though far from using the full capabilities of GENDACS, this test (Figure 9) illustrated how an engineer can read data of one type, reorder it, and write it out as a different type.

```
IN:Field#      Type          Conversion Instructions      OUT:Field#      Type
 1.............integer........>.............................6........real
 2.............real...........>.............................2........real
 3.............shortint.......>.............................5........real
 4.............byte...........>.............................1........real
 5.............longint........>.............................4........real
 6.............word...........>.............................3........real
```

Figure 10. Changing Data Order and Type.

Note the reordering of the numbers under `OUT:Field#`. Resultant data are written out in the numerical order given in the `OUT:Field#` list.

C. *Actual Data Conversion*

For a genuine test of this project, GENDACS was given an actual data file collected from field instruments by SDL. This file held several megabytes of data. Each block of data within the file consisted of 1,280 16-bit "word" values. The first three data values contained a time signature indicating when the data were obtained. The first three lines of Figure 3 demonstrate how these time values were decoded into fractional hours. The rest of the block held actual measurements that needed to be scaled and processed with calibration values. After a few minutes, the data conversion was complete. Review of the results showed that GENDACS indeed processed the data correctly.

V. RECOMMENDATIONS

Originally, it was intended for GENDACS to also process data files stored in variable length ASCII form. It became apparent however, that designing the program to process fixed length, binary files would be sufficient to satisfy the scope of this senior project. As a suggestion, perhaps future updates of GENDACS would include support for ASCII input data files. The original proposal also suggested other options that may be added to GENDACS in the future:

Command-line Execution. The ability to run this program from the command prompt will allow for batch file processing of multiple files.

Extended Data Analysis. This may include reporting maximum/minimum values and other brief statistical assessments of the data.

Data Compression. This would allow compressed input files to be processed. The data would be decompressed temporarily and then run through the program.

VI. CONCLUSION

Though completing the project GENDACS became a complex and laborious task, it certainly turned out to be worthwhile. There were a number of design goals to this project. The primary goal was to create a data processing software package that would be versatile enough to process nearly any type of data file. Created for the MS-DOS operating system, GENDACS succeeded in processing 8-bit, 16-bit, and 32-bit signed and unsigned types of data. The program also successfully processed the 6-byte floating-point data type "real". GENDACS does not contain predefined file configurations that limit it to specific input file formats. Instead, at run time, the user describes the input file and the program adapts to be able to process it. The program does allow the user to save the file description and reload it at another time.

Another design goal of this project dealt with memory usage. Working with very large data files made it necessary to efficiently manage memory. GENDACS does just that. Data structures are created dynamically and occupy only as much memory as necessary at any instant. Also, because GENDACS works with individual blocks of data at one time as opposed to the entire file at once, sufficient memory is generally available.

In addition, the importance of choosing a good programming language became apparent. Written in the object-oriented programming (OOP) language Turbo Pascal 6.0, this project should readily allow for updates to the original code. The OOP style enforces a high degree of modularity and creates code that is easily extensible. It is a great improvement over the standard style of programming. As new data types may be created, changes to the program to accommodate these new types are more easily made.

Overall, most of the design goals were fulfilled. This project served as a superior teaching tool in learning the object-oriented style of programming. It also helped in understanding how data are collected by digital field equipment and why it must be processed for further analysis. Most importantly, it is anticipated that the program GENDACS will be used in a wide variety of data processing applications. In that lies the true value of this project.

Appendix

Conversion Instruction Set

Conversion String Codes

- Using immediate values

`+N`	Add N to current input value.
`-N`	Subtract N from current input value.
`*N`	Multiply current input value by N.
`/N`	Divide current input value by N.
`N-`	Subtract current input value from N.
`N/`	Divide N by current input value.

- Using n^{th} input values

`+in`	Add value at input location "n" to current input value.
`-in`	Subtract value at input location "n" from current input value.
`*in`	Multiply current input value by value at input location "n".
`/in`	Divide current input value by value at input location "n".
`in-`	Subtract current input value from value at input location "n".
`in/`	Divide value at input location "n" by current input value.

- Using n^{th} output values

`+on`	Add value at output location "n" to current input value.
`-on`	Subtract value at output location "n" from current input value.
`*on`	Multiply current input value by value at output location "n".
`/on`	Divide current input value by value at output location "n".
`on-`	Subtract current input value from value at output location "n".
`on/`	Divide value at output location "n" by current input value.

- Special codes

`>`	No Conversion. Carry number straight across.
`x`	Skip this input data value.

Note: Each conversion instruction on the same line must be separated by a semicolon.

Vitae

Kevan L. Morgan

1499 Ellendale Avenue
Logan, Utah 84321-2842
(801) 752-5825

EDUCATION **Utah State University, Logan, Utah**

Majors: *Electrical Engineering*, Bachelor of Science, June 1993
Liberal Arts and Sciences, Bachelor of Arts, June 1993
with Departmental Honors in Electrical Engineering
and University Honors

Minors: *History* and *Portuguese*

HONORS & AWARDS

Utah State University Club Scholar, 1986
Jack & Bonnie Parson Scholarship, College of Engineering, 1986
Frederick P. Champ Scholarship, Boy Scouts of America, 1986
Larry S. Cole Scholarship, College of Engineering, 1992
Tau Beta Pi, National Engineering Honors Society, 1990-93
Deans List, 1989-93
Eagle Scout, 1981

ACTIVITIES

President, USU Honors Program Student Council, 1992-93
USU Honors Council Member - Interactions and Peer Advising, 1989-92
Engineering Council Freshman Representative, 1986
Associated Students of USU Fall Leadership Retreat, 1991
Honors Program Fall Retreat Facilitator, 1992
Portuguese Club, 1991-93
University Investment Club, 1990-92
Church Representative in Portugal, 1987-89

WORK EXPERIENCE

Space Dynamics Laboratory, Utah State University., April 1991 - present
Software Engineer

- Wrote computer programs in Turbo Pascal for data extraction
- Installed a large software package for Sun (Unix) workstations
- Completed an engineering senior project (lasted 8 months)

Bourns Networks, Inc., Logan, Utah., June 1990 - February 1991
Engineering Aide

- Directed experiments for new quality assurance tests
- Designed computer interface for testing equipment

Early Intervention Research Institute,
Department of Psychology, Utah State University., October 1989 - June 1990
Research and Office Clerk